CENG 242
Homework # 5

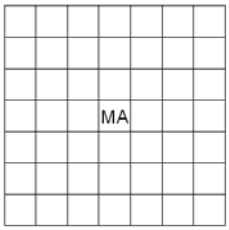
(Due: May 1st, 2006 Monday 23:59)

You should have learned Symbol Table in previous homeworks. Now it is time to play some game ☺

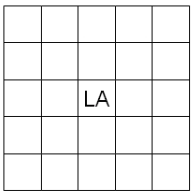
First, let's define the game. The game is a turn based board arcade game. There are two teams. One is **Attacker**, the other is **Defender**. Each team has three members: "Major", "Lieutenant" and "Private", where Major has more skills than Lieutenant and Lieutenant has more skills than Private. One of the members of the Defender team has flag. The mission of the Attacker team is to get the flag from Defender team and the mission of the Defender team is to keep the flag until predetermined number of turns pass. In each turn, first, Major Attacker does his move, then Major Defender, then Lieutenant Attacker, then Lieutenant Defender, then Private Attacker and lastly Private Defender does his move and a turn finishes. After finishing a turn, next turn begins. And game continues until an Attacker moves to the square where flag exist and hold by no Defender (in this case Attacker team wins) or predetermined number of turns have passed and Attackers could still not get the flag (in this case Defender team wins).

Now, define some terms used above.

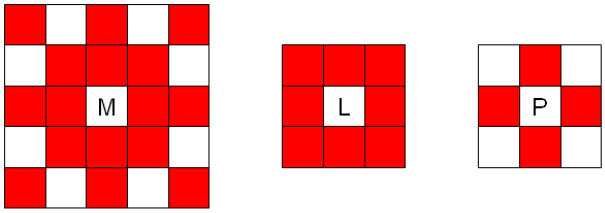
Board: It can be thought as chess board, but has size *n* x *n*.
Move of an Attacker: It is either walking to a square in his act area or shooting to a Defender in his act area. Each Attacker have gun and predetermined number of ammo. If he is out of ammo he could not shoot any more.
Move of a Defender: It is either walking to a square in his act area or raising his shield to protect one of his sides (north, south, east or west). If he is raising the shield and an attacker shot him from that direction (e.g. if he is raising the shield against north and attacker shot from north, northwest or northeast; if he is raising the shield against west and attacker shot from west, northwest or southwest etc.) he drops his shield and he can not raise shield (anyway he can do nothing) until his move. If he is shot from unprotected side, he dies and drops the flag if he has.
Vision Area: The players can not see the whole board; they just can see a limited part of the board. A major can see 7x7 squares of the board, where he is assumed to be in the middle:



The others can see 5x5:



Act Area: The players can walk and shoot to a limited area. The act areas are colored with red.



Major can walk or shoot to a square with distance two as shown above, if there is no other player in between.

Now, let's define your task. There are 3 classes: Attacker, Defender and GameEngine. GameEngine class will be given you and you will implement the other two.

```
enum MoveTypeAttacker {WALKA, SHOOT};
enum MoveTypeDefender {WALKD, CHANGESHIELDDIRECTION,
RAISESHIELD};
enum Direction {NORTH, SOUTH, EAST, WEST, NORTHEAST, NORTHWEST,
SOUTHEAST, SOUTHWEST};
enum ObjectType {MAJOR, LIEUTENANT, PRIVATE, FLAG, WALL};
enum TeamType {ATTACKER, DEFENDER};

struct Objects {
        int x; //relative position
        int y;
        TeamType tt;
        ObjectType ot;
        bool hasFlag; // for defenders
        bool hasShield; // for defenders
        Direction dirShield; // direction of shield, for defenders
        bool hasAmmo; // for attackers
};

struct Move {
        bool isValid;
        union {
                MoveTypeAttacker attAct;
                MoveTypeDefender defAct;
```

```
        };
        Direction dir;
        unsigned dist;
};

class Attacker {
  private:
        unsigned numAmmo;
        Objects * visionArea;
        unsigned sizeVisionArea;

  public:
        Attacker(unsigned hasAmmo) {numAmmo = hasAmmo;}
        virtual ~Attacker();
        Move move(Objects * visA, unsigned sizeVisA) { sizeVisionArea = sizeVisA;
visionArea = visA; Move tm; tm = shoot(); if (tm.isValid) return tm; return walk(); }
        virtual Move walk() = 0;
        virtual Move shoot() = 0;
};

class MajorAttacker:public Attacker {
  private:

  public:
        MajorAttacker(unsigned hasAmmo):Attacker(hasAmmo) {}
        ~MajorAttacker();
        Move walk();
        Move shoot();
};

class LieutenantAttacker:public Attacker {
  private:

  public:
        LieutenantAttacker(unsigned hasAmmo):Attacker(hasAmmo) {}
        ~LieutenantAttacker();
        Move walk();
        Move shoot();
};

class PrivateAttacker:public Attacker {
  private:

  public:
        PrivateAttacker(unsigned hasAmmo):Attacker(hasAmmo) {}
        ~PrivateAttacker();
        Move walk();
        Move shoot();
```

```
};


class Defender {
  private:
        unsigned numShield;
        Direction shieldDirection;
        bool raisingShield;
        bool flag;
        bool dead;
        Objects * visionArea;
        unsigned sizeVisionArea;

  public:
        Defender(unsigned numS, bool flg) {numShield = numS; flag = flg; raisingShield
= false; dead = false; }
        virtual ~Defender();
        Move move(Objects * visA, unsigned sizeVisA) { sizeVisionArea = sizeVisA;
visionArea = visA; Move tm; tm = walk(); if (tm.isValid) return tm; return raiseShield();
}
        virtual Move walk() = 0;
        virtual Move raiseShield() = 0;
        bool hasFlag() {return flag;}
        void setFlag() {flag=true;}
        bool isDead() {return dead;}
        bool isRaisingShield() {return raisingShield;}
        void gotHit() {numShield--;raisingShield=false;}
        void die() {dead = true;}
        Direction getShieldDirection() {return shieldDirection;}
};

class MajorDefender:public Defender {
  private:

  public:
        MajorDefender(unsigned numS, bool flg):Defender(numS, flg) {}
        ~MajorDefender();
        Move walk();
        Move raiseShield();
};

class LieutenantDefender:public Defender {
  private:

  public:
        LieutenantDefender(unsigned numS, bool flg):Defender(numS, flg) {}
        ~LieutenantDefender();
        Move walk();
```

```
        Move raiseShield();
};

class PrivateDefender:public Defender {
  private:

  public:
        PrivateDefender(unsigned numS, bool flg):Defender(numS, flg) {}
        ~PrivateDefender();
        Move walk();
        Move raiseShield();
};
```

You will implement unimplemented functions. **walk**, **raiseShield** and **shoot** will be as follows:

shoot and walk for Attacker:
If flag is unattended (dropped by a dead defender) walk to the flag if you can (which finishes the game and attackers win), else get as close as you can to it.
shoot:
You can shoot to any square in the act area if there is no other player in between.
If you have no ammo left return a Move typed value with isValid=false.
If there exist Defenders in your act area, shoot the most important one. Importance is defined as follows:
*A defender unprotected from your side is more important then protected one
*A defender with Flag > Major > Lieutenant > Private
And return a Move typed value with isValid=true
If no Defender exist in your act area, return a Move typed value with isValid=false

walk for Attacker:
Find the most important Defender in your vision area and get close as you can to him.
If no Defender in your vision area exists, move randomly

raiseShield and walk for Defender:
You can raise the shield in four directions (N, W, E, S).
If you have no shield left return a Move typed value with isValid=false for raiseShield function. If an Attacker has no ammo left, do not consider him as Attacker, he can not do anything to you☺ If flag is unattended (dropped by a dead defender) in your act area, walk to it if you can, else get as close as you can to it. For each Attacker in your vision area, detect if you are in act area of him. If you are in act area of some Attacker find a square in your act area which is not an act area of any of Attackers. If there are more then one such square, find the square farthest to most powerful attacker (distance is Manhattan distance). If there are more then one farthest, choose one of them randomly. If there is no square which is not an act area of any of Attackers, raise your shield to the side of most powerful attacker (Power order: Major > Lieutenant > Private). If you are not in act area of any attacker, move to the farthest position to the most powerful attacker that cannot be attacked by any player. If no such position exists, raise the shield against the most

powerful attacker. If no players exist in your vision area, move randomly.

The last task is to take a look at GameEngine.h and GameEngine.cpp to understand how Game Engine works.